contains no more domain elements than those named by the constant symbols. Under the resulting semantics, which we call database semantics to distinguish it from the standard semantics of first-order logic, the sentence Equation (8.3) does indeed state that Richard’s two brothers are John and Geoffrey. Database semantics is also used in logic programming systems, as explained in Section 9A.5.

It is instructive to consider the set of all possible models under database semantics for the same case as shown in Figure 8.4 Figure 8.5 shows some of the models, ranging from the model with no tuples satisfying the relation to the model with all tuples satisfying the relation. With two objects, there are four possible two-element tuples, so there are $2^4 = 16$ different subsets of tuples that can satisfy the relation. Thus, there are 16 possible models in all—a lot fewer than the infinitely many models for the standard first-order semantics. On the other hand, the database semantics requires definite knowledge of what the world contains.

This example brings up an important point: there is no one “correct” semantics for logic. The usefulness of any proposed semantics depends on how concise and intuitive it makes the expression of the kinds of knowledge we want to write down, and on how easy and natural it is to develop the corresponding rules of inference. Database semantics is most useful when we are certain about the identity of all the objects described in the knowledge base and when we have all the facts at hand; in other cases, it is quite awkward. For the rest of this chapter, we assume the standard semantics while noting instances in which this choice leads to cumbersome expressions.

### 8.3 USING FIRST-ORDER LOGIC

Now that we have defined an expressive logical language, it is time to learn how to use it. The best way to do this is through examples. We have seen some simple sentences illustrating the various aspects of logical syntax; in this section, we provide more systematic representations of some simple domains. In knowledge representation, a domain is just some part of the world about which we wish to express some knowledge.

We begin with a brief description of the TELL/ASK interface for first-order knowledge bases. Then we look at the domains of family relationships, numbers, sets, and lists, and at
the wumpus world. The next section contains a more substantial example (electronic circuits) and Chapter 12 covers everything in the universe.

8.3.1 Assertions and queries in first-order logic

Sentences are added to a knowledge base using TELL, exactly as in propositional logic. Such sentences are called assertions. For example, we can assert that John is a king, Richard is a person, and all kings are persons:

```assert
TELL(KB, King(John)) .
TELL(KB, Person(Richard)) .
TELL(KB, \( \forall x \text{ King}(x) \rightarrow \text{Person}(x) \)) .
```

We can ask questions of the knowledge base using ASK. For example,

```query
ASK(KB, King(John))
```
returns true. Questions asked with ASK are called queries or goals. Generally speaking, any query that is logically entailed by the knowledge base should be answered affirmatively. For example, given the two preceding assertions, the query

```query
ASK(KB, Person(John))
```
should also return true. We can ask quantified queries, such as

```query
ASK(KB, \( \forall x \text{ Person}(x) \))
```

The answer is true, but this is perhaps not as helpful as we would like. It is rather like answering "Can you tell me the time?" with "Yes." If we want to know what value of \( x \) makes the sentence true, we will need a different function, ASK VARS, which we call with

```query
ASK VARS(KB, Person(x))
```
and which yields a stream of answers. In this case there will be two answers: \( \{x/\text{John}\} \) and \( \{x/\text{Richard}\} \). Such an answer is called a substitution or binding list. ASK VARS is usually reserved for knowledge bases consisting solely of Horn clauses, because in such knowledge bases every way of making the query true will bind the variables to specific values. That is not the case with first-order logic; if \( KB \) has been told \( \text{King}(\text{John}) \lor \text{King}(\text{Richard}) \), then there is no binding to \( x \) for the query \( \forall x \text{ King}(x) \), even though the query is true.

8.3.2 The kinship domain

The first example we consider is the domain of family relationships, or kinship. This domain includes facts such as "Elizabeth is the mother of Charles" and "Charles is the father of William" and rules such as "One's grandmother is the mother of one's parent."

Clearly, the objects in our domain are people. We have two unary predicates, Male and Female. Kinship relations—parenthood, brotherhood, marriage, and so on—are represented by binary predicates: Parent, Sibling, Brother, Sister, Child, Daughter, Son, Spouse, Wife, Husband, Grandparent, Grandchild, Cousin, Aunt, and Uncle. We use functions for Mother and Father because every person has exactly one of each of these (at least according to nature's design).